دانشگاه اصفهان
دانشکده مهندسی کامپیوتر

مستند پروژه اول (بهترین پرواز)

درس مبانی و کاربردهای هوش مصنوعی

دکتر کارشناس

اعضای گروه:
متین اعظمی (۴۰۰۳۶۲۳۰۰۳)
امیرعلی لطفی (۴۰۰۳۶۱۳۰۵۳)
زهرا معصومی (۴۰۰۳۶۲۳۰۳۴)

پاییز ۱۴۰۲

خواندن و آماده‌سازی داده‌ها

در ابتدا فایل csv با دستور زیر خوانده می‌شود:

```
flights = pd.read_csv('Dataset.csv')
```

سپس داده‌های تکراری با استفاده از تابع drop_duplicates حذف می‌گردند.

برای راحتی کار، مقادیر ستون‌های مختصات هر فرودگاه را به یک ستون جدید به صورت یک tuple با نام‌های SourceAirport_Location و DestinationAirport_Location تبدیل شده است.[1]

شرح ستون‌های data frame مورد استفاده به صورت زیر خواهد بود:

- Airline: نام شرکت هواپیمایی
- SourceAirport: نام فرودگاه مبدا
- DestinationAirport: نام فرودگاه مقصد
- SourceAirport_City: شهر فرودگاه مبدا
- SourceAirport_Country: کشور فرودگاه مبدا
- DestinationAirport_City: شهر فرودگاه مقصد
- DestinationAirport_Country: کشور فرودگاه مقصد
- Distance: فاصله بین دو فرودگاه
- FlyTime: مدت زمان پرواز
- Price: هزینه پرواز
- SourceAirport_Location: مختصات جغرافیایی فرودگاه مبدا
- DestinationAirport_Location: مختصات جغرافیایی فرودگاه مقصد

[1] این تاپل به ترتیب طول جغرافیایی، عرض جغرافیایی و ارتفاع از سطح دریا را مشخص می‌کند.

گراف حالت

در این قسمت گراف حالت مسئله از روی data frame داده شده ساخته می‌شود. برای این‌کار نیاز است که دو کلاس زیر تعریف شوند:

کلاس Node

در گراف حالت، هر فرودگاه به عنوان یک گره در نظر گرفته می‌شود. به همین علت یک کلاس گره تعریف می‌شود که اطلاعات فرودگاه را در خود ذخیره می‌کند.

```
class Node:
    def __init__(self, airport, city, country, coordination):
        self.airport = airport
        self.city = city
        self.country = country
        self.coordination = coordination
```

کلاس Edge

اگر بین دو فرودگاه یک سفر وجود داشته باشد، در گراف حالت، بین گره آن دو فرودگاه یک یال در نظر گرفته می‌شود. اطلاعات مربوط به این سفر نیز در این کلاس ذخیره می‌شود.

همچنین در این کلاس، یک تابع به نام `get_score` تعریف شده است. این تابع یک ترکیب خطی از سه مقدار `distance`، `fly_time` و `price` را، با ضرایب دلخواه، محاسبه می‌کند و برمی‌گرداند.
ضرایب این ترکیب خطی هر کدام مقدار پیش فرضی دارند که به شرح زیر می‌باشد:

- ضریب مقدار `distance`: `w1` با مقدار پیش فرض `1`
- ضریب مقدار `fly_time`: `w2` با مقدار پیش فرض `700`
- ضریب مقدار `price`: `w3` با مقدار پیش فرض `2`

مقدار خروجی این تابع به عنوان هزینه این یال استفاده می‌شود.

```python
class Edge:
    def __init__(self, airline, distance, fly_time, price):
        self.airline = airline
        self.distance = distance
        self.fly_time = fly_time
        self.price = price

    def get_score(self, w1 = 1, w2 = 700, w3 = 2):
        return self.distance*w1 + self.fly_time*w2 + self.price*w3
```

ذخیره‌سازی گراف

برای ذخیره‌سازی گراف از دو دیکشنری[2] با نام‌های nodes و graph استفاده می‌شود.

دیکشنری nodes

کلیدهای nodes، اسامی فرودگاه‌ها و مقدار آن‌ها شی‌ای از کلاس Node است که فرودگاهی با نام کلید نظیرش را تعریف می‌کند. این سبک پیاده‌سازی و استفاده از دیکشنری به عنوان ساختمان‌داده، به ما کمک می‌کند تا شی مورد نظر را تنها با استفاده از نام فرودگاه با پیچیدگی زمانی $O(1)$ به دست آوریم.

دیکشنری graph

ذخیره‌سازی گراف در این دیکشنری به صورت adjacency map صورت می‌گیرد. به این معنی که هر کلید نام یک گره است و مقدار آن یک دیکشنری دیگر است که گره‌های همسایه آن گره را ذخیره می‌کند؛ بدین صورت که نام گره همسایه به عنوان کلید و یک شی از کلاس Edge به عنوان مقدار این کلید ذخیره می‌شود؛ که ارتباط بین دو گره را مشخص می‌کند.

[2] dictionary

الگوریتم مقداردهی دو دیکشنری مذکور به شکل زیر است:

```
for data in df.values:
    airline, src_airport, dest_airport, src_city, src_country,
dest_city, dest_country, distance, fly_time, price, src_loc, dest_loc
= data
    if src_airport not in nodes:
        nodes[src_airport] = Node(src_airport, src_city, src_country,
src_loc)

    if dest_airport not in nodes:
        nodes[dest_airport] = Node(dest_airport, dest_city,
dest_country, dest_loc)

    e = Edge(airline, distance, fly_time, price)

    if src_airport not in graph:
        graph[src_airport] = {}

    if dest_airport not in graph:
        graph[dest_airport] = {}

    graph[src_airport][dest_airport] = e
```

پیاده‌سازی الگوریتم Dijkstra

از آن‌جایی که یال‌های گراف هیچ‌وقت منفی نخواهند بود، برای یافتن کوتاه‌ترین مسیر بین دو گره در گراف حالت، الگوریتم دایکسترا[3] بر روی گراف اجرا می‌شود. تفاوت این حالت با حالتی که این الگوریتم به صورت درختی استفاده می‌شود این است که در این حالت پیچیدگی زمانی الگوریتم $O(nlogn)$ خواهد بود ولی در حالت اجرای درختی این پیچیدگی نمایی خواهد شد.

داده‌ساختارها

صف اولویت pq

در این الگوریتم از داده‌ساختار heap یا priority queue استفاده می‌شود. برای این‌کار از کتابخانه heapq در پایتون استفاده شده است. این داده‌ساختار وظیفه ذخیره‌سازی هر گره به همراه فاصله‌ای که تا آن گره محاسبه شده است را دارد. هر عضو این داده‌ساختار به صورت یک tuple ذخیره شده است که عضو اول آن فاصله محاسبه شده و عضو دوم آن نام فرودگاه است. در نتیجه داده‌ساختار heap، المان‌ها را بر اساس عضو اول tuple، یعنی فاصله محاسبه شده برمی‌گرداند.

دیکشنری parent

همچنین برای ذخیره‌سازی مسیر طی شده، از یک دیکشنری به نام `parent` استفاده شده است. در این دیکشنری هر کلید نام یک فرودگاه است و مقدار نظیر آن، نام فرودگاهی است که به عنوان گره بعدی فرودگاه کلید، در مسیر انتخاب شده است.

دیکشنری dist

این دیکشنری کمترین هزینه رسیدن به هر فرودگاه را مشخص می‌کند. کلیدهای این دیکشنری نام فرودگاه‌ها و مقدار نظیر آن‌ها، بهترین هزینه‌ای است که می‌توان از گره ابتدایی به آن رسید.

[3] Dijkstra

الگوریتم

در این الگوریتم، یک صف اولویت وجود دارد که در هر تکرار از حلقه یک المان از آن خارج می‌شود و حلقه تا زمانی ادامه دارد که این صف خالی نشده باشد.
هر المان خارج شده، یک tuple است. عضو اول این tuple که d نام‌گذاری شده است، هزینه مسیر طی شده تا آن گره است؛ و عضو دوم آن که u نام‌گذاری شده است، نام فرودگاه آن گره است.
سپس به ازای هر گره همسایه گره u، که v نام‌گذاری می‌شود، هزینه رسیدن از گره u به v محاسبه می‌شود. این هزینه با استفاده از تابع get_score که در کلاس Edge تعریف شده است، به دست می‌آید. یال بین دو گره u و v را می‌توان از دیکشنری graph به دست آورد. هزینه به دست آمده برای یال weight نامیده می‌شود.
سپس بررسی می‌شود که آیا قبلا گره v دیده شده است یا خیر. اگر گره v قبلا دیده نشده بود، یا با هزینه بیشتری نسبت به هزینه کنونی دیده شده بود، هزینه رسیدن به گره v برابر با هزینه رسیدن به گره u به علاوه weight قرار داده می‌شود.
سپس این مقدار به همراه نام گره، در صف اولویت قرار داده می‌شود. ترتیب این دو مقدار در قسمت "داده‌ساختارها - صف اولویت pq" شرح داده شده است.
همچنین دیکشنری parent را نیز به‌روزرسانی می‌کنیم.
این کار آنقدر تکرار می‌شود که دیگر صف pq خالی شود. یعنی کوتاه‌ترین مسیر به هر گره پیدا شده باشد.

```
while pq:
            d, u = heap.heappop(pq)
            for v in graph[u]:
                weight = graph[u][v].get_score()
                if v not in dist:
                    dist[v] = dist[u] + weight
                    heap.heappush(pq, (dist[v], v))
                    parent[v] = u

                elif dist[v] > dist[u] + weight:
                    dist[v] = dist[u] + weight
                    heap.heappush(pq, (dist[v], v))
                    parent[v] = u
```

اجرای یک نمونه

با فراخوانی تابع زیر:

```
dijkstra('Imam Khomeini International Airport', 'Raleigh Durham International Airport')
```

و با پردازش روی خروجی این تابع، خروجی زیر را می‌توان مشاهده کرد:

Dijkstra

Exe time: 0.09381747245788574s

.-

Flight #1 (Iran Air):

From: Imam Khomeini International Airport - Tehran, Iran

To: Amsterdam Airport Schiphol - Amsterdam, Netherlands

Duration: 4075.580681472172km

Time: 5.562833144685928km

Price: 2007.790340736086km

Flight #2 (easyJet):

From: Amsterdam Airport Schiphol - Amsterdam, Netherlands

To: Newcastle Airport - Newcastle, United Kingdom

Duration: 522.06486620626km

Time: 1.1485277841071553km

Price: 231.03243310313km

Flight #3 (Jetstar Airways):

From: Newcastle Airport - Newcastle, United Kingdom

To: Melbourne International Airport - Melbourne, Australia

Duration: 834.6449978671836km

Time: 1.5368260843070605km

Price: 387.3224989335918km

Flight #4 (American Airlines):

From: Melbourne International Airport - Melbourne, Australia

To: Charlotte Douglas International Airport - Charlotte, United States

Duration: 791.2302148525258km

Time: 1.482894676835436km

Price: 365.6151074262629km

Flight #5 (American Airlines):

From: Charlotte Douglas International Airport - Charlotte, United States

To: Raleigh Durham International Airport - Raleigh-durham, United States

Duration: 208.5119182907748km

Time: 0.7590210165102793km

Price: 74.25595914538741km

Total Price: $3066.016339344458

Total Duration: 6432.032678688916km

Total Time: 10.49010270644586h

برای یافتن مسیر طی شده، از دیکشنری parent و تابع get_path_details استفاده می‌کنیم.

تابع یافتن مسیر

برای پیدا کردن مسیر طی شده، باید روی گره‌های دیده شده بازگردیم. برای اینکار نیاز است که از گره پایانی یعنی dest شروع کنیم و هر بار گره‌ای که از آن به این گره رسیده‌ایم را پیدا کنیم. اگر این مسیر در آرایه‌ای ذخیره شود، مسیر برگشت مشخص می‌شود. با وارونه کردن المان این آرایه، مسیر به درستی قابل دسترسی خواهد بود.

پیاده‌سازی الگوریتم A*

این الگوریتم برای امتیازدهی به هر گره حالت، از دو مقدار استفاده می‌کند. تابع امتیازدهی هر گره به شکل زیر است:

$$f(n) = g(n) + h(n)$$

در این تابع از دو مقدار $g(n)$ و $h(n)$ استفاده می‌شود. مقدار $g(n)$ همان هزینه مسیر تا به الان طی شده است. تابع $h(n)$ یک تابع اکتشافی است که به ما کمک می‌کند از میان گره‌های درون لیست مقدم[4]، کدام گره را به عنوان گره بعدی انتخاب کنیم.

تابع اکتشافی

برای این مسئله، تابع اکتشافی در نظر گرفته شده، از فاصله دو فرودگاه روی کره استفاده شده است. این فاصله با استفاده از فرمول هاورسین[5] محاسبه می‌گردد. این فرمول به شرح زیر است:

$$a = sin^2(\frac{lat_2 - lat_1}{2}) + cost(lat_1).cos(lat_2).sin^2(\frac{lon_2 - lon_1}{2})$$

$$c = 2.atan2(\sqrt{a}, \sqrt{1-a})$$

$$distance = R.c + |alt_2 - alt_1|$$

که lat_i و lon_i طول و عرض جغرافیایی و alt_i ارتفاع از سطح دریای نقطه i می‌باشد؛ به ازای $i = 1, 2$. و R شعاع کره می‌باشد (شعاع کره زمین ۶,۳۷۱,۰۰۰ متر است).

داده‌ساختارها

صف اولویت pq

همانند همان صف اولویت در الگوریتم دایکسترا می‌باشد با این تفاوت که هر المان این صف یک tuple است که سه عضو دارد. عضو اول همان مقدار $f(n)$ می‌باشد. زیرا که می‌خواهیم صف اولویت المان‌ها را با در نظر گرفتن مقدار تابع اکتشافی مرتب کند. عضو دوم فقط مقدار $g(n)$ است. زیرا این مقدار در محاسبه $f(n)$ گره‌های همسایه استفاده می‌شود. عضو سوم نیز نام گره است.

[4] frontier

[5] Haversine

دیکشنری parent و dist

این دو دیکشنری همانند دو دیکشنری مورد استفاده در الگوریتم دایکسترا است که در آن قسمت توضیح داده شده است.

الگوریتم

در این الگوریتم، یک صف اولویت وجود دارد - که همان لیست مقدم است - و در هر تکرار حلقه یک المان از آن خارج می‌شود و حلقه تا زمانی ادامه دارد که این صف خالی نشده باشد.
هر عضو خارج شده از صف اولویت یک tuple است که اعضای آن در قسمت "داده‌ساختارها - صف اولویت pq" شرح داده شده است.
بعد از خارج کردن یک عضو از صف اولویت، آزمون هدف انجام می‌شود و بررسی می‌شود که آیا گره خارج شده همان گره حالت هدف است یا خیر. در صورت مثبت بودن این آزمون پرچم found مقدار درست می‌گیرد و حلقه شکسته می‌شود.
در غیراین‌صورت، حلقه‌ای روی همه‌ی گره‌های همسایه‌ی این گره اجرا می‌شود (گره همسایه v نامیده می‌شود). در هر تکرار این حلقه، مقدار $g(n)$ که همان score است و $h(n)$ که همان h است محاسبه می‌شود.
اگر گره v هیچ‌گاه مشاهده نشده بود و یا با هزینه بیشتری مشاهده شده بود، هزینه رسیدن به آن و دیکشنری parent را به‌روزرسانی می‌کنیم.
همچنین یک tuple جدید با مقادیر $(f(n),\ g(n),\ v)$ به صف اولویت pq اضافه می‌شود.

```
while pq:
        s, d, u = heap.heappop(pq)

        # Goal Test
        if u == dest:
            found = True
            break

        for v in graph[u]:
                e = graph[u][v]
                h = heuristic(v, dest)
                score = d + e.get_score()
                if (v not in dist) or (v in dist and dist[v] >=
```

```
score):
                    dist[v] = score
                    parent[v] = u
                    heap.heappush(pq, (score + h, score, v))
```

اجرای یک نمونه

با فراخوانی تابع زیر:

```
a_star('Imam Khomeini International Airport', 'Raleigh Durham
International Airport')
```

و با پردازش روی خروجی این تابع، خروجی زیر را می‌توان مشاهده کرد:

```
A*
Exe time: 0.1081550121307373s
.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-
Flight #1 (Mahan Air):
From: Imam Khomeini International Airport - Tehran, Iran
To: Düsseldorf Airport - Duesseldorf, Germany
Duration: 3921.8990130989446km
Time: 5.450957619833174km
Price: 457.8679576636519km
----------------------------
Flight #2 (Lufthansa):
From: Düsseldorf Airport - Duesseldorf, Germany
To: Newcastle Airport - Newcastle, United Kingdom
Duration: 700.4016904718744km
Time: 0.9686450823247468km
Price: 162.00341803579576km
----------------------------
Flight #3 (Jetstar Airways):
From: Newcastle Airport - Newcastle, United Kingdom
To: Melbourne International Airport - Melbourne, Australia
Duration: 834.6449978671836km
Time: 1.4205257067621924km
Price: 117.1949201700933km
----------------------------
Flight #4 (American Airlines):
From: Melbourne International Airport - Melbourne, Australia
To: Charlotte Douglas International Airport - Charlotte, United States
Duration: 791.2302148525258km
Time: 1.6713531473383905km
Price: 144.40649101693947km
```

```
----------------------------
Flight #5 (American Airlines):
From: Charlotte Douglas International Airport - Charlotte, United States
To: Raleigh Durham International Airport - Raleigh-durham, United States
Duration: 208.5119182907748km
Time: 0.898608088101398km
Price: 97.2285064411616km
----------------------------
Total Price: $978.7012933276421
Total Duration: 6456.687834581304km
Total Time: 10.410089644359902h
```

کتابخانه‌های مورد استفاده

- numpy
- pandas
- matplotlib

منابع

- دوره هوش‌مصنوعی دانشگاه شریف (استاد رهبان)